

HoGent

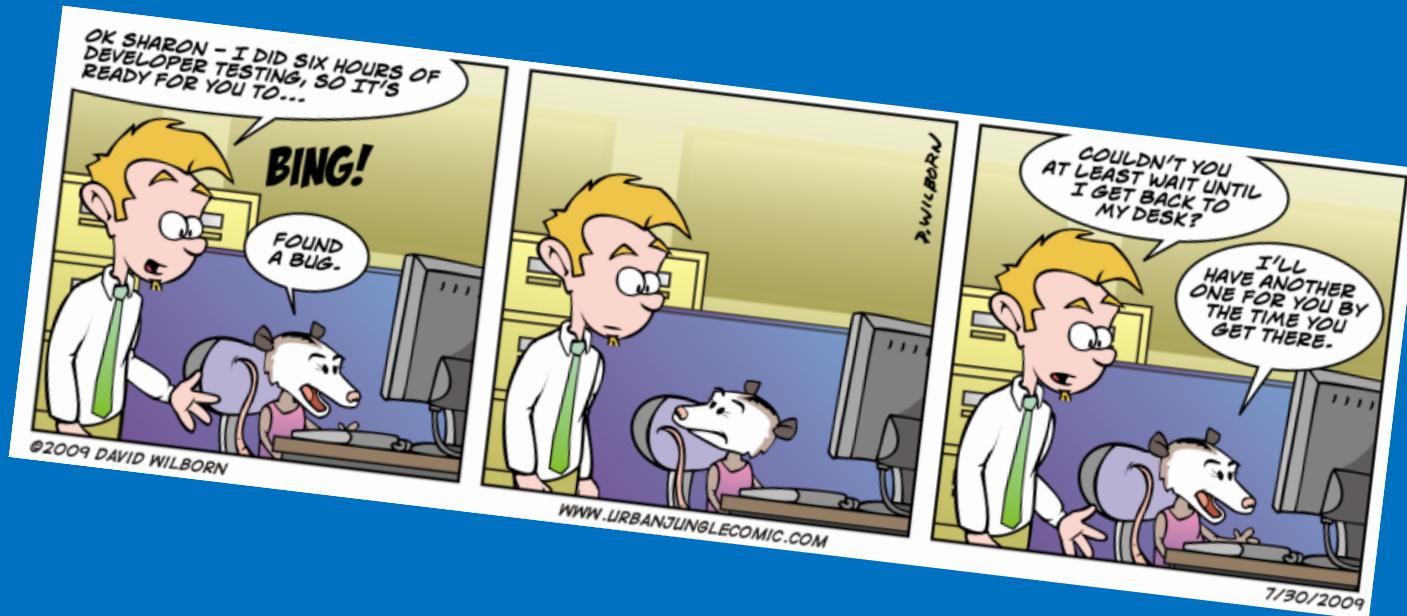
BEDRIJF
EN
ORGANISATIE

Hoofdstuk 4 : Debugging

Debugging

1. Inleiding
2. Debugging
3. Referenties

Inleiding



HoGent

1. Inleiding

▶ Mogelijke fouten

- **Syntax fouten**

Een programma met syntax fouten **kan je niet uitvoeren**. Je moet syntax fouten verbeteren alvorens je je programma kan uitvoeren

- **Logische fouten**
- **Run time fouten**

Deze fouten **komen voor tijdens de uitvoering** van een programma. Het opsporen en verbeteren van deze fouten is **debugging**.

Clone de applicatie <https://github.com/WebIII/4thDebugging>

1. Inleiding

▶ Syntax fouten

- Fouten tegen de programmeertaal
- Ontdekt at design-time of bij compilatie
 - ; vergeten, code die niet gedeclareerde variabelen gebruikt, statische type-check errors,
...
- Worden rood onderlijnd weergegeven + tooltip (met muis erover)
- Staan opgeliist in de **Error List**
 - menu View > Error List
 - na Build > Build solution: een **succesvolle build** is enkel mogelijk als je code **geen syntax fouten** meer bevat

1. Inleiding

▶ Syntax fouten: Error-List

- Voor meer info over fout **lees je de description!** De boodschap is meestal vrij duidelijk en vertelt wat de fout is ☺

Je kan de lijst van getoonde errors op
meerdere manieren filteren

Error List					
Entire Solution		Code	Description	Project	File
		CS1002	; expected	Banking..NETCoreApp, Version=v1.0	BankAccount.cs
		CS0664	Literal of type double cannot be implicitly converted to type 'decimal'; use an 'M' suffix to create a literal of this type	Banking..NETCoreApp, Version=v1.0	SavingsAccount.cs
		CS0506	'SavingsAccount.Withdraw(decimal)': cannot override inherited member 'BankAccount.Withdraw(decimal)' because it is not marked virtual, abstract, or override	Banking..NETCoreApp, Version=v1.0	SavingsAccount.cs
		CS0664	Literal of type double cannot be implicitly converted to type 'decimal'; use an 'M' suffix to create a literal of this type	Banking..NETCoreApp, Version=v1.0	SavingsAccount.cs

Klik op de Code om on-line
hulp in te schakelen

Dubbelklik op de fout om naar
de lijn code te gaan waar de
error zich voordeet

Los de fouten op!

1. Inleiding

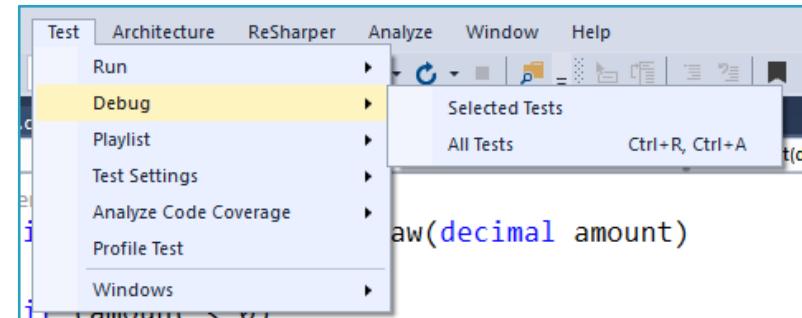
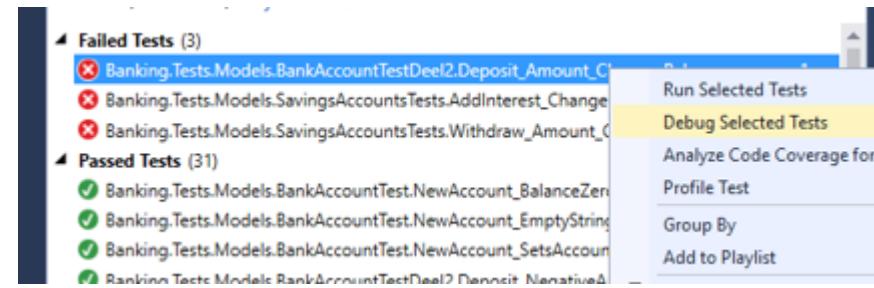
▶ Logische fouten

- Programma wordt uitgevoerd maar werkt niet zoals verwacht
- Een logische fout kan je niet altijd ontdekken
 - ze kunnen verborgen blijven tot de code op een bepaalde manier wordt gebruikt
- Foutopsporing via **debuggen** laat tijdelijke onderbreking van de uitvoering van het programma toe
 - De programma flow te bekijken : bepalen welke methodes werden aangeroepen
 - Lijn per lijn door de code te gaan
 - De waarden van variabelen, properties en uitdrukkingen te bekijken en eventueel aan te passen
 - Methodes uit te testen
 - ...
- Starten Debug proces: menu Debug > Start Debugging, of F5

1. Inleiding

▶ Logische fouten

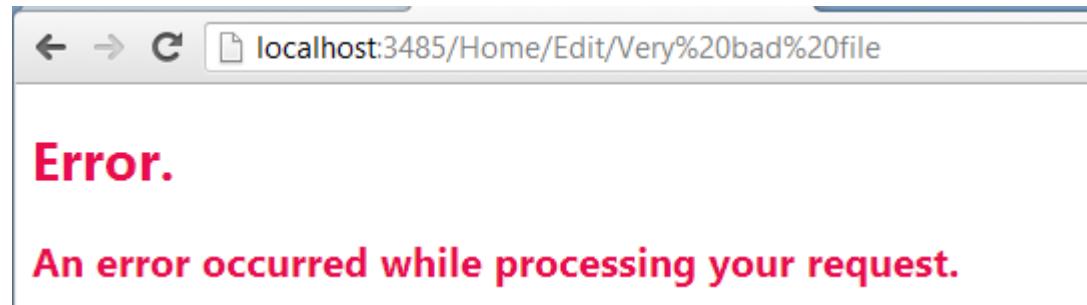
- **Unit testing** is de belangrijkste manier om het aantal logische fouten in je code te minimaliseren
- Tijdens het unit testen wordt je code uitgevoerd
 - je kunt via de debugger deze uitvoering tijdelijk onderbreken...
 - menu Test > Debug, of rechtermuisklik op test in Test Explorer > Debug test



1. Inleiding

▶ Run time fouten

- De uitvoering van het programma wordt gestopt door een fout -> Probeer dit via foutafhandeling op te vangen!
 - In debug mode wordt het programma gestopt waar de fout zich voordoet.
 - Anders wordt omgeleid naar Error page



- Dit wordt later in de cursus behandeld.

Debugging



HoGent

2. Debugging

- ▶ Run de unit testen
 - De onderstaande testen falen

Failed Tests (3)		
✖	Banking.Tests.Models.BankAccountTestDeel2.Deposit_Amount_ChangesBalance	77 ms
✖	Banking.Tests.Models.SavingsAccountsTests.AddInterest_ChangesBalance	81 ms
✖	Banking.Tests.Models.SavingsAccountsTests.Withdraw_Amount_CausesTwoTransactions	3 ms

- We maken gebruik van debugging om te fouten op te lossen

2. Debugging

- ▶ <http://www.codeproject.com/Articles/79508/Mastering-Debugging-in-Visual-Studio-2010-A-Beginn>
- ▶ Code lijn per lijn uitvoeren om te zien wat er gebeurt
- ▶ Maak gebruik van **breakpoints**
 - Een breakpoint = een regel waarbij de uitvoering van het programma tijdelijk onderbroken wordt
 - Klik de Margin Indicator bar naast de lijn met code of selecteer de code en druk F9 (rode bol)
 - Voorbeeld
 - De eerste unit test die niet slaagt heeft te maken met het storten (deposit) van geld op je rekening.
 - Plaats breakpoint 1ste lijn in methode Deposit in Bankaccount.
 - Start de test in debugging mode: uitvoering stopt bij die regel (gele pijl). Deze lijn moet nog worden uitgevoerd!
 - Om het verloop van de code verder te bestuderen. Gebruik menu Debug, of de Debug toolbar

2. Debugging

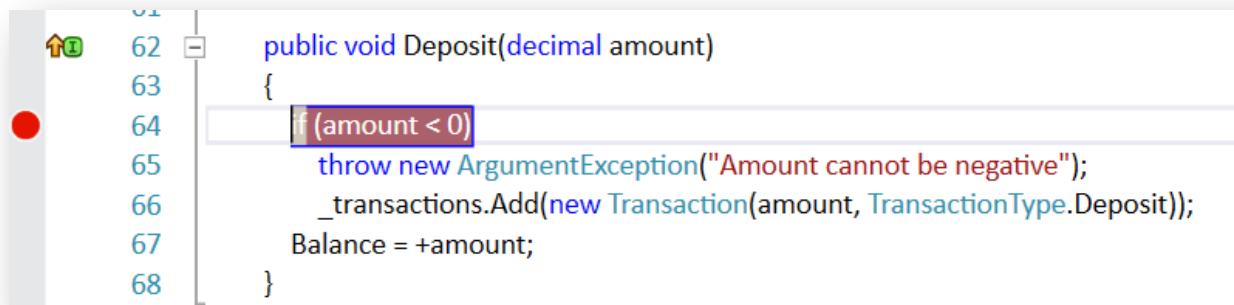
- ▶ Commando's beschikbaar in break mode (Debug menu/Toolbar)
 - **Start/Continue** (F5, Debug > Start) : programma (verder) uitvoeren tot volgende breakpoint of tot einde
 - **Break all** : onderbreek programma onmiddellijk en ga naar instructie die wordt uitgevoerd
 - **Stop** : stop de uitvoering en exit de debugger (Shift + F5, Debug > Stop Debugging)
 - **Restart** : stop uitvoering en start programma opnieuw (Ctrl+Shift+F5, Debug > Restart)
 - **Toon volgende instructie** : ga naar de volgende instructie die zal worden uitgevoerd
 - **Step into** (F11, Debug > Step into) : voer volgende lijn code uit. Is de volgende lijn een methode, voer eerste instructie methode uit.
 - **Step over** (F10, Debug > Step Over) : voer volgende lijn code uit. Is dit een methode, voer de methode volledig uit als zijnde 1 instructie
 - **Step out** (Shift+F11, Debug > Step out) : voer de rest van de instructies in de huidige methode uit en onderbreek als dit gedaan is.



2. Debugging

▶ Voorbeeld

- Eénmaal in break mode



A screenshot of a debugger interface showing a code editor with the following code:

```
public void Deposit(decimal amount)
{
    if (amount < 0)
        throw new ArgumentException("Amount cannot be negative");
    _transactions.Add(new Transaction(amount, TransactionType.Deposit));
    Balance = +amount;
}
```

The line `if (amount < 0)` is highlighted with a purple rectangle, indicating it is the current line of execution. A red circular breakpoint marker is visible on the left margin next to line 64.

- Druk F11, om de lijn code uit te voeren.
- Druk een aantal maal op F11 : voert aangeduide lijn uit, ...
- Zie hoe je via F11 uiteindelijk in de code de constructor Transaction springt
- Druk vervolgens Shift + F11 : de methode wordt volledig uitgevoerd, ...
- Druk F11
- Druk F5 : programma voert verder uit tot volgend breakpoint/einde

2. Debugging

- ▶ De debugging windows (Debug > Windows)
 - De breakmode wordt vaak gebruikt om **waarden van variabelen of expressies te bekijken**, en de veranderingen die de waarde ondergaat, op te volgen. Dit kan op verschillende manieren:
 - Plaats **muisaanwijzer** boven de variabelenaam.
 - **Quick watch** : toont de waarde van een variabele, uitdrukking
 - **Autos Window** : toont de namen van de variabelen, waarden en gegevenstypes van huidig statement en de voorgaande coderegel
 - **Local Window** : tonen/aanpassen van waarden variabelen in huidige procedure/klasse (locale variabelen)
 - **Watch window** : om de waarde van een variabele, .. te volgen en aan te passen. Toont naam, waarde en gegevenstype
 - Of om code uit te testen
 - **Immediate window**

2. Debugging

- **DataTips:** met **muisaanwijzer** over variabele
 - **Voorbeeld**
 - Via F10 komen we aan de regel `Balance =+ amount`

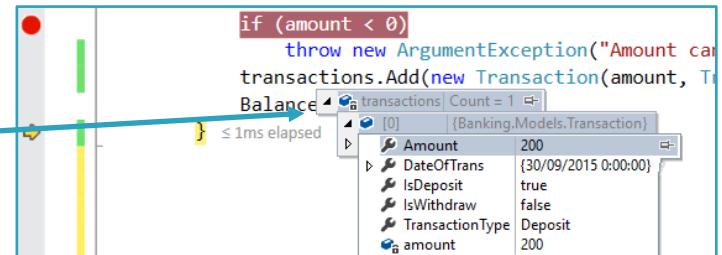
The screenshot shows a code editor with a red breakpoint at line 62. The cursor is at line 67, where the code is `Balance =+ amount;`. A yellow DataTip box is open, showing the value `Balance | 0`. The status bar indicates `≤ 2ms elapsed`.

Als we de muisaanwijzer boven Balance plaatsen krijgen we de waarde te zien in een DataTip

Klik op de pin om de DataTip het scherm te houden (ook via rechtermuisklik op Balance > Pin to source), zie debug menu voor beheer van DataTips

Bij complexere types kan je inzoomen op het gewenste niveau van detail, ga bv. met muisaanwijzer over transactions en klap open...

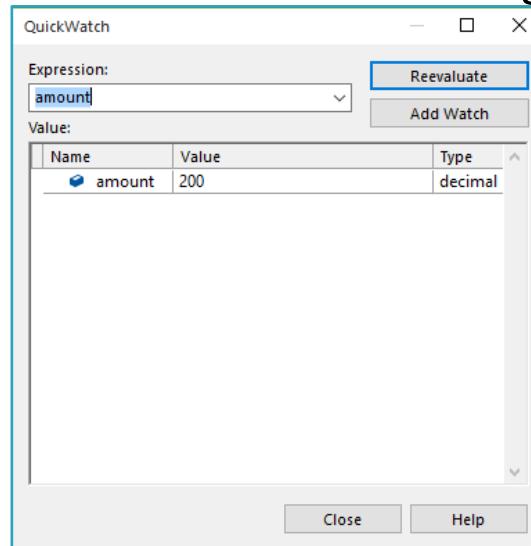
Druk op ctrl-toets om de code op de voorgrond te brengen...



2. Debugging

- De Quick Watch

- Laat je toe **variabelen en expressies te bekijken en te evalueren**
- Selecteer variabele en druk CTRL+ALT+Q of kies Quick Watch uit context sensitief menu
- Quick Watch toont 1 variabele/expressie tegelijkertijd
- Je moet het venster sluiten om verder te kunnen debuggen
 - Je kan van hieruit wel een variabele toevoegen aan het Watch window



2. Debugging

◦ Variable Windows

- om variabelen en expressies te bekijken, te evalueren en te editeren
- voor elke variabele/expressie zie je naam, type en de waarde
- 3 soorten

Locals	Autos	Watch
Wordt automatisch gevuld door de debugger met variabelen binnen de huidige context of scope (~methode)	Wordt automatisch gevuld door de debugger met variabelen in de huidige code-regel en in de voorgaande code-regel	Hier bepaal je zelf welke variabelen/expressies je wilt toevoegen

2. Debugging

Locals

Wordt automatisch gevuld door de debugger met variabelen binnen de huidige context of scope (~methode)

- Voorbeeld Locals Window:

- Debug > Windows > Locals of CTRL+ALT+V,L

The screenshot shows the Visual Studio IDE interface. In the top right corner, there is a teal-colored box containing the word "Locals". Below this, a tooltip-like box provides a definition: "Wordt automatisch gevuld door de debugger met variabelen binnen de huidige context of scope (~methode)".

The main area shows a code editor with the following C# code:

```
public void Deposit(decimal amount)
{
    if (amount < 0)
        throw new ArgumentException("Amount cannot be negative");
    _transactions.Add(new Transaction(amount, TransactionType.Deposit));
    Balance += amount;
}
```

A red circle icon is visible on the left side of the code editor. A yellow arrow points from the bottom-left towards the Locals window.

Below the code editor, a "Locals" window is open, displaying the following table:

Name	Value	Type
decimal.operator < returned	false	bool
this	{123-4567890-02 - 0}	Banking.Models.BankA
amount	200	decimal

2. Debugging

- Voorbeeld Autos Window:

- Debug > Windows > Autos or CTRL+ALT+V,A

Autos

Wordt automatisch gevuld door de debugger met variabelen in de **huidige code-regel** en in de **voorgaande code-regel**

The screenshot shows the Visual Studio IDE with the Autos window open. On the left, there's a vertical toolbar with icons for Up, Down, Breakpoint, and Step. The code editor shows a C# method named Deposit with several lines highlighted in yellow, indicating they are part of the current or previous code execution. The Autos window at the bottom lists local variables with their names, values, and types.

```
public void Deposit(decimal amount)
{
    if (amount < 0)
        throw new ArgumentException("Amount cannot be negative");
    _transactions.Add(new Transaction(amount, TransactionType.Deposit));
    Balance += amount;
}
```

The screenshot shows the Visual Studio IDE with the Autos window open. A blue arrow points from the line of code in the editor to the corresponding variable entry in the Autos window. The Autos window displays the following information:

Name	Value	Type
decimal.operator < returned	false	bool
TransactionType.Deposit	Deposit	Banking.Models.TransactionType
_transactions	Count = 0	System.Collections.Generic.IList<Banking.Models.Transaction>
amount	200	decimal
this	{123-4567890-02 - 0}	Banking.Models.BankAccount

2. Debugging

Watch

Hier bepaal je **zelf** welke variabelen/expressies je wilt **toevoegen**

- **Voorbeeld Watch window:**

- Debug > Windows > Watch
- Rechtermuisklik op variabele, selecteer in context menu Add Watch of via Debug > Windows > Watch (4 vensters)

Name	Value	Type
Balance	200	decimal
amount	200	decimal

- Voorbeeld
 - Voeg Balance en amount toe aan het Watch window
 - Controleer of de waarde telkens verandert (kleurt rood bij verandering) als je door de code stapt met F11. Controleer startwaarde en eindwaarde
 - Voeg eventueel ook andere expressies toe. Zoek oorzaak van de foute Balance. Pas code aan en controleer opnieuw.

2. Debugging

◦ Het Watch window

- Elke instructie die geldig is binnen een programma, werkt binnen een watch venster
 - Voorbeeld : System.Threading.Thread.Sleep(2000) -> programma zal 2 seconden wachten (zie hourglass). Zal dan “Expression has been evaluated and has no value” teruggeven om aan te geven dat de instructie toch werd uitgevoerd maar geen return waarde heeft
- Zoals bij de andere variabele windows kan je ook de waarde van een variabele aanpassen...
 - Voorbeeld
 - Vervang de waarde van Balance, plaats deze op 300. Wat gebeurt er?

Watch 1		
Name	Value	Type
Balance	300	decimal
amount	200	decimal
_transactions	Count = 1 {Banking.Models.Transaction}	System.Collections.Generic.IList<Banking.Models.Trans Banking.Models.Transaction
[0]		
Raw View		

2. Debugging

▶ Het venster **Immediate**

- Kent 2 modi
 - **Command** : intypen van Visual Studio opdrachten
 - SaveAll, Close, Help, Exit,...
 - **Immediate** : Invoeren van instructies terwijl het programma onderbroken is
 - Waarde opvragen van variabele, property of uitdrukking: ?
 - Waarde aanpassen van variabele of property
 - Procedures, methodes uitvoeren



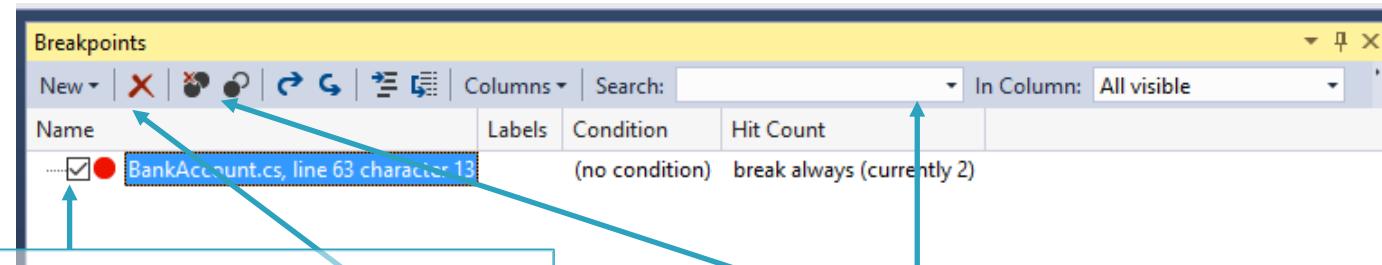
The screenshot shows the Immediate Window in Visual Studio. The window title is "Immediate Window". The content displays the following variable values:

```
?Balance
300
?_transactions
Count = 1
[0]: {Banking.Models.Transaction}
?amount
200
```

2. Debugging

▶ Breakpoints – geavanceerde opties

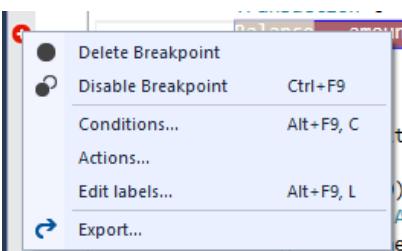
- Debug > Windows > Breakpoints
 - Beheer van breakpoints: verwijderen, toevoegen, enable/disable



Je kan een breakpoint tijdelijk
disablen/enablen. Wanneer een
breakpoint disabled is, wordt het
genegeerd tijdens het debuggen.

disable/enable of verwijderen van
breakpoints die voldoen aan de
zoekcriteria

verwijderen van geselecteerde breakpoints



enable/disable van een breakpoint kan
ook via **toolbar**,
of via **context sensitief menu...**

The code editor shows a method named 'Deposit'. A red circular breakpoint icon is positioned next to the line number 63. The code on line 63 contains an if-statement that throws an exception if the amount is less than zero.

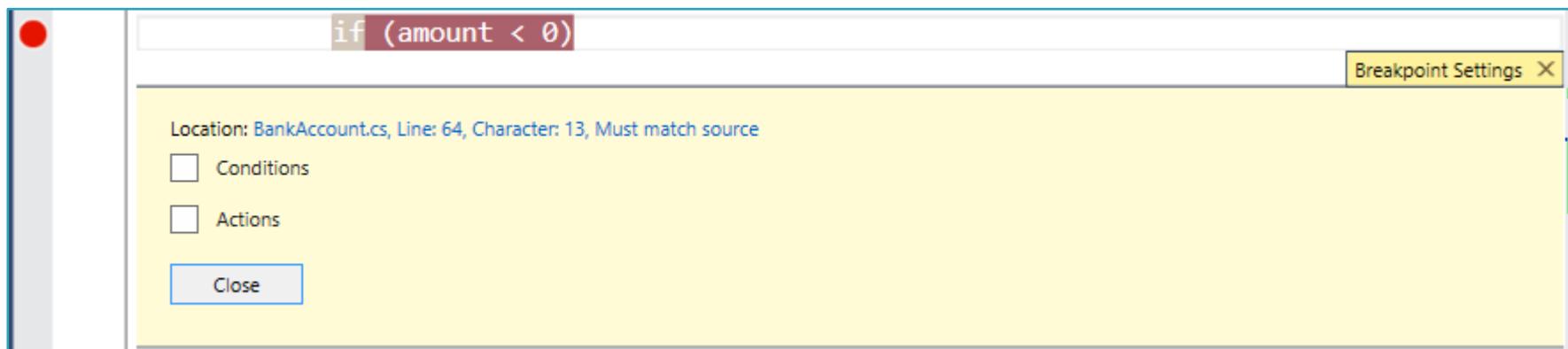
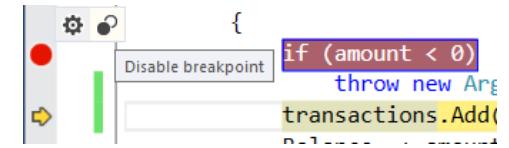
```
public void Deposit(de
{
    if (amount < 0)
        throw new Argum
    _transactions.Add(n
    Balance += amount;
}
```

2. Debugging

▶ Breakpoints – geavanceerde opties

◦ Settings

- rechtermuisklik op breakpoint in breakpoints venster
- of toolbar, of context sensitief menu
 - dan verschijnen settings in een peek window, dit is handig in gebruik

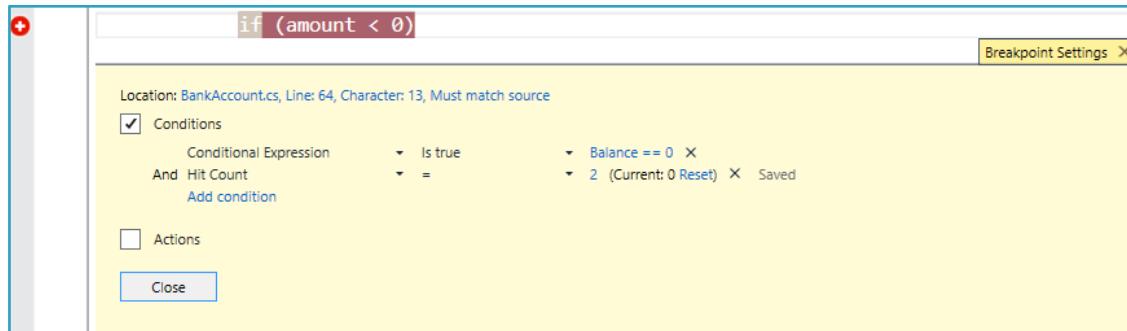


2. Debugging

▶ Breakpoints – geavanceerde opties

- **Settings - Conditions:**

- **conditional expression:** breakpoints zullen een onderbreking in de code veroorzaken wanneer er aan een bepaalde conditie is voldaan, of wanneer een bepaalde conditie is veranderd...
- **hit count:** breakpoints zullen een onderbreking in de code veroorzaken wanneer ze een hit count bereiken, of een veelvoud van een hit count bereiken, of boven een bepaalde hitcount gaan...



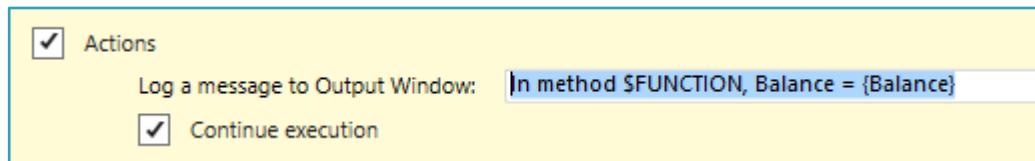
Dit breakpoint zal actief zijn wanneer er een tweede keer langs wordt gepasseerd,
én Balance == 0

2. Debugging

▶ Breakpoints – geavanceerde opties

- **Settings - Actions:**

- **log message:** gebruik vrije tekst, variabelen (gebruik {}), of voorgedefinieerde systeemvariabelen (gebruik \$) om een bericht naar het output window te sturen
 - je kan instellen of uitvoering moet doorgaan of moet onderbroken worden
 - merk op hoe alles steeds werkt met IntelliSense...



In output window na het passeren van de breakpoint

In method Banking.Models.BankAccount.Withdraw(decimal), Balance = 200

2. Debugging

▶ PerfTips

- Geven informatie over de performantie (~tijd) van je code
 - wanneer je stap per stap door de code loopt
 - of van breakpoint tot breakpoint

```
public IEnumerable<Transaction> GetTransactions(DateTime? from, DateTime? till)
{
    if (from == null && till == null) return transactions;
    if (from == null) from = DateTime.MinValue;
    if (!till.HasValue) till = DateTime.MaxValue;
    IList<Transaction> transList = new List<Transaction>();
    foreach (Transaction t in transactions)
        if (t.DateOfTrans >= from && t.DateOfTrans <= till)
            transList.Add(t);
    return transList;
}
```

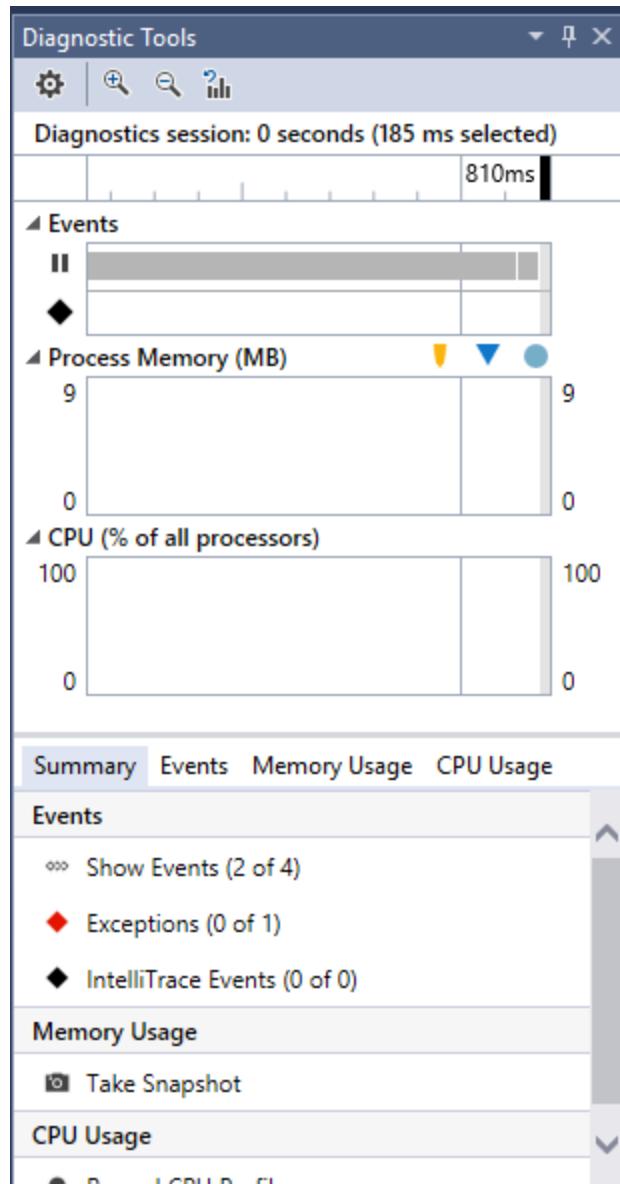
≤ 1ms elapsed

PerfTip toont hoe lang het duurt om deze methode uit te voeren (d.i. van eerste breakpoint tot tweede), klik op een PerfTip om de Diagnostic Tools te openen...

2. Debugging

▶ Diagnostic tools

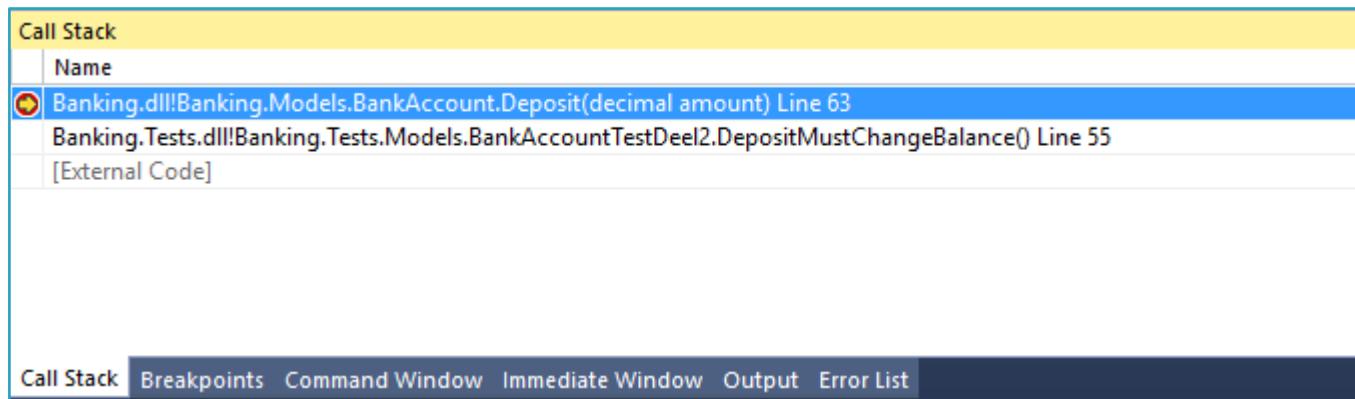
- Debug > Windows > Show Diagnostic tools
- Meer gedetailleerde info over je debug sessie
 - Voor elke stap die tijdens de debugsessie genomen wordt kan je analyseren wat impact was op geheugen/tijd
 - Je ziet hoeveel exceptions er reeds opgevangen zijn



2. Debugging

▶ Call stack

- Volgen van de flow van programma (info over elke functieaanroep)
- Ctrl+Alt+C of Debug > Windows > Call Stack



2. Debugging

▶ Run to click debugging

- Kies Debug > Step Into. De code stopt voor de uitvoering van de eerste lijn code of plaats een breakpoint en klik F5
- Hover over de lijn waar je een volgend breakpoint wil plaatsen. Er verschijnt een groene pijl “Run execution to here”. Klik op de groene pijl en de code wordt tot daar uitgevoerd. Ook nu kan je de variabelen bekijken.

The screenshot shows a code editor with the following C# code:

```
static void Main(string[] args)
{
    namespace Banking
    {
        class Program
        {
            static void Main(string[] args)
            {
                BankAccount("123-4567890-02");
                tNumber: {account.AccountNumber} ");
                e: {account.Balance} ");

                account.Deposit(200M);
            }
        }
    }
}
```

A green arrow icon labeled "Run execution to here" is positioned next to the closing brace of the inner Main method. A tooltip above the cursor shows the same code with the arrow icon.

- Selecteer bvb Deposit > Rechtsklik > Step into Specific en dan wordt de code uitgevoerd tot de eerste lijn in deze methode.
- Hier kan je ook werken met “Run execution to here”

2. Debugging

▶ Debug.WriteLine(if) commando

- Debug class is onderdeel van **System.Diagnostics**.
- Print een boodschap naar de Visual Studio console.
- De uitvoering ervan gebeurt in debug mode (je hoeft de instructies dus niet te verwijderen bij release!)
 - merk op dat je wel je code bezoedelt met extra statements
- Tijdens runnen kan je Output venster bekijken (View > Output Window).

```
public void Deposit(decimal amount)
{
    if (amount < 0)
        throw new ArgumentException("Amount cannot be negative");
    Debug.WriteLine($"Balance = {Balance}");
    _transactions.Add(new Transaction(amount, TransactionType.Deposit));
    Balance += amount;
}
```

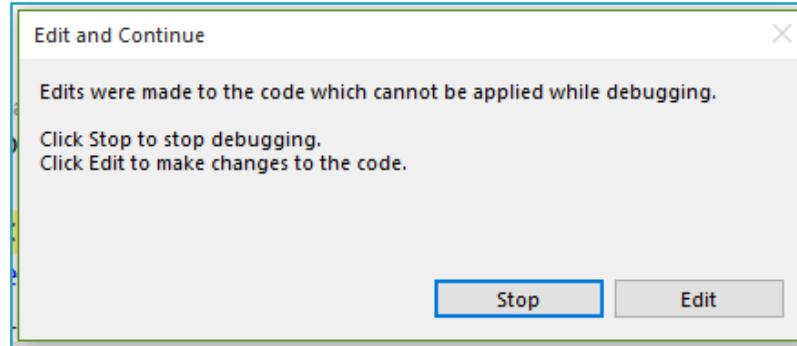
Output

```
Show output from: Debug
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.0.1\System.Text.Encoding.Extensions.dll'
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.0.1\System.Threading.Tasks.dll'
'dotnet.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\1.0.1\System.Diagnostics.Debug.dll'
Balance = 0
```

2. Debugging

▶ Edit and continue

- Als de debugger stopt bij breakpoint, kan je code aanpassen en verdergaan met debuggen. VS hercompileert de applicatie en hercreëert de status van je applicatie op het debugger break moment



- Deze optie is standaard enabled
 - Enable/Disable:
 - Project > Properties > Web > onderaan in Debuggers sectie “Enable Edit and Continue”
 - Run applicatie en ga ergens in breakmode. Voeg een lijn code toe (bvb var i=1;) en ga verder met de uitvoering

2. Debugging

- ▶ Nog dieper gaan:
 - Script explorer : debuggen van scripts
 - Threads
 - Toont threads igv multi threaded applicaties
 - Modules
 - Toont de dll's en exe's gebruikt door het project
 - Memory venster
 - Toont stukje geheugen gebruikt door applicatie
 - Disassembly
 - Toont de assembly code die overeenkomt native code gecreëerd door de Just-in-Time (JIT) compiler, (niet de MSIL code gegenereerd door de Visual Studio compiler.)
 - Registers
 - Toont de inhoud van de registers

The screenshot shows the Microsoft Visual Studio Disassembly window. The assembly code is displayed in a list format with addresses, opcodes, and comments. The code is for a method named 'Deposit(decimal)' in 'Banking.Models.BankAccount'. The assembly instructions include calls to 'Banking.Models.BankAccount.Deposit(decimal)', various mov and lea instructions involving memory pointers and registers (eax, edx, esp), and a sub instruction. The disassembly window has tabs for 'Disassembly', 'Banking', 'Transaction.cs', and 'SavingsAc'.

```
Disassembly  □ × Banking Transaction.cs SavingsAc
Address: Banking.Models.BankAccount.Deposit(decimal)
Viewing Options
06B3CF2E call    dword ptr ds:[63A0398h]
06B3CF34 nop
06B3CF35 mov     eax,dword ptr [ebp-3Ch]
06B3CF38 mov     dword ptr [ebp-70h],eax
06B3CF3B lea     edx,[ebp-80h]
06B3CF3E mov     ecx,dword ptr [ebp-3Ch]
06B3CF41 call    06B1E0F8
06B3CF46 lea     eax,[ebp-80h]
06B3CF49 sub     esp,10h
06B3CF4C movq   xmm0,mmword ptr [eax]
06B3CF50 movq   mmword ptr [esp],xmm0
06B3CF55 movq   xmm0,mmword ptr [eax+8]
06B3CE5A mova   mmword ptr [esp+8],xmm0
```

2. Debugging

▶ Voorbeeld

- Withdraw_Amount_CausesTwoTransactions Test van de SavingAccount faalt nog steeds.
 - Dit is een logische fout: de SavingAccount zou 3 transacties moeten bevatten en bevat er maar 1.
 - Wat is de reden?
 - In de code voor Withdraw bij SavingsAccount zien we niet een direct een probleem.
 - We kunnen de fout isoleren door de aanroep naar Deposit te volgen.
 - In Deposit wordt er wel een transactie aangemaakt maar blijkbaar is daar later geen spoor meer van.
 - De logica van Deposit klopt niet...
 - De testen voor Deposit zijn onvolledig ☹ : een deel van de te verwachten logica wordt niet getest...
- Oplossing : TDD
 - 1. Schrijf test die logische fout controleert bvb Withdraw_Amount_AddsTransaction
 - 2. Test moet falen (nu kan je debuggen om de fout te vinden)
 - 3. Pas de code aan
 - 4. Test moet slagen

2. Debugging

- ▶ Het is belangrijk dat je alle gedrag test. Schrijven van testen doet je nadenken over het gedrag van een methode.
- ▶ Ook fouten die door klanten of test team worden doorgegeven of nieuwe functionaliteiten, dien je op een TDD aan te pakken.

2. Debugging

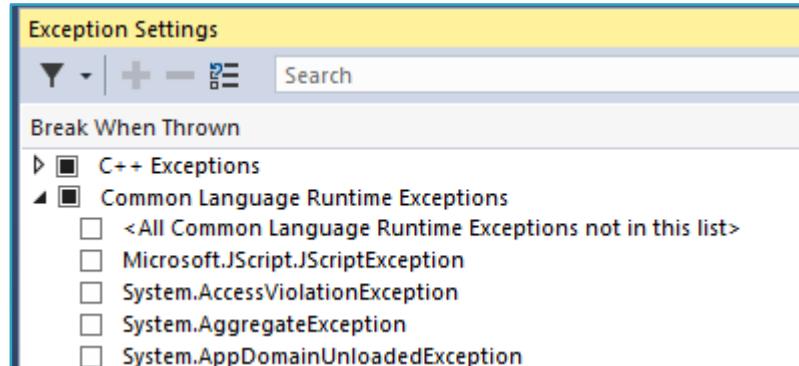
- ▶ Debug configuratie
 - Gebeurt automatisch bij eerste maal klik op Debug. De web.config wordt aangemaakt met

```
<compilation debug="true"/>
```
 - Afzetten in productie!

2. Debugging

▶ Debug mode en exceptions

- Normaliter : Het programma wordt onderbroken enkel bij unhandled exceptions. De debugger stopt in dit geval bij de lijn code waar exception zich voordoet. De debugger stopt niet bij afgehandelde exceptions.
- Kan je aanpassen : menu Debug > Exceptions Settings
- In het **Exceptions Settings** venster kan je aangeven bij welke CLR exceptions je een break wenst



- De lijn code waar de exception zich voordoet (of enkele regels ervoor) is een goed startpunt voor debugging
- Voeg onderstaande code

2. Debugging

▶ Debug mode en exceptions

- Als de exception niet wordt opgevangen, wordt de uitvoering van de code onderbroken
 - Voeg in de methode Main onderaan volgende code toe en run
 - savingsAccount.Withdraw(1000M);
 - De code wordt onderbroken en de melding wordt getoond

A screenshot of a debugger interface showing a code editor and a call stack window. The code editor displays a C# method with a yellow highlight over the line 'throw new InvalidOperationException("Balance cannot be negative");'. A tooltip from the call stack window shows the exception details: 'Exception Unhandled' and 'System.InvalidOperationException: 'Balance cannot be negative''. Below the tooltip, there are links for 'View Details' and 'Copy Details'.

```
public override void Withdraw(decimal amount)
{
    if (amount + WithdrawCost > Balance)
        throw new InvalidOperationException("Balance cannot be negative");
    base.Withdraw(amount);
    base.Withdraw(
}
1 reference | 0 changes |
```

```
public void AddInterest()
{
    Deposit(Balance * interestRate);
}
```

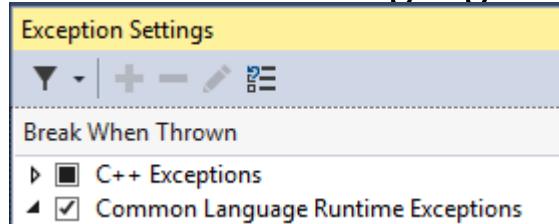
2. Debugging

▶ Debug mode en exceptions

- Vang je de exception op, dan wordt het runnen van de code niet onderbroken.
 - Pas de code als volgt aan en run opnieuw

```
try
{
    savingsAccount.Withdraw(1000M);
}
catch (Exception ex)
{
    Console.WriteLine("ERROR " + ex.Message);
}
Console.ReadKey();
```

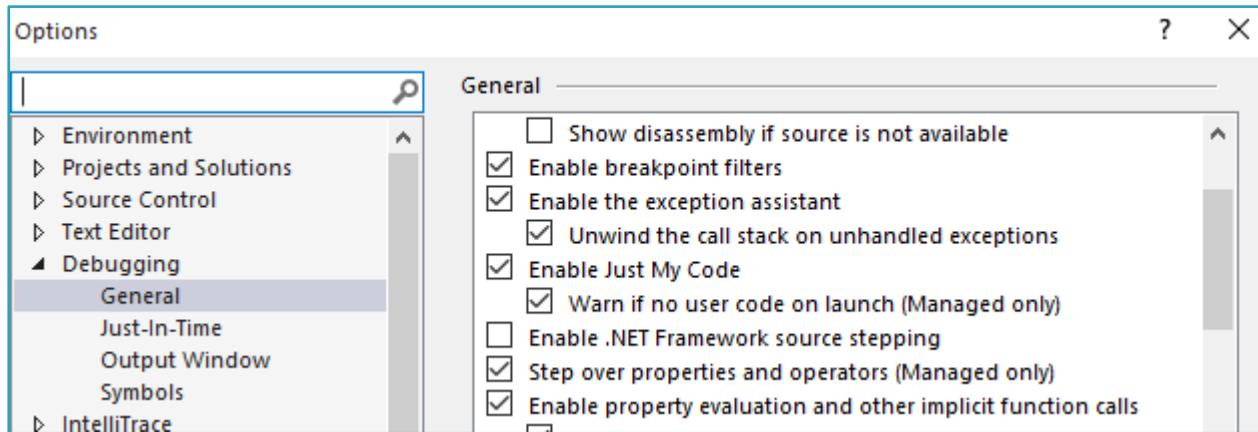
- In de Diagnostic Tools kan je in de Summary tab zien dat er een Exception geworpen is. Als je op de tab Events klikt kan je de detail van de Exception bekijken
- Wens je toch te stoppen bij elke exception die geworpen wordt: Debug > Window > Exception Settings en vink Common Language Runtime Exceptions aan



2. Debugging

▶ Debug mode en exceptions

- Normaliter : Programma zal ook stoppen bij exceptions in gegenereerde code
- Kan je aanpassen : “enable or disable Just My Code debugging”
 - In Tools menu, kies Options.
 - In Options dialog box, open Debugging node en kies General.
 - Vink Enable Just My Code aan of uit.



2. Debugging

▶ IntelliTrace

- Beschikbaar in Visual Studio 2015 Enterprise edition
- Advanced debugging : Met IntelliTrace neem je de acties op en kan je dit later terug bekijken, afspelen
- Hiermee kan je ook data in productie verzamelen
- Meer op
 - <http://blogs.msdn.com/b/visualstudioalm/archive/2015/01/16/intellitrace-in-visual-studio-ultimate-2015.aspx>
 - <http://channel9.msdn.com/Events/TechEd/NorthAmerica/2012/DEV365>
 - <https://channel9.msdn.com/Series/Visual-Studio-2012-Premium-and-Ultimate-Overview-FRA/IntelliTrace-Experience-in-Visual-Studio-2015-FRA>

Referenties

Referenties

- ▶ Debugging in Visual Studio :
<https://msdn.microsoft.com/en-us/library/sc65sadd.aspx>
- ▶ Pluralsight: Visual Studio, a first look at the IDE – Debugging Improvements
<http://www.pluralsight.com/courses/visual-studio-2015-first-look-ide>
- ▶ Mastering Debugging in Visual Studio - A Beginner's Guide :
<http://www.codeproject.com/Articles/79508/Mastering-Debugging-in-Visual-Studio-2010-A-Beginn>